Monday Trade
Advancing the Future of Open Finance

June 24, 2025

1 Introduction

Monday Trade is a decentralized exchange protocol engineered for high-throughput, low-latency trading in fully onchain environments. It features a hybrid execution model that integrates discrete order book logic with range-based AMM liquidity, allowing for precise capital allocation and efficient price formation.

Central to the system is a dual tick architecture: fine-grained order ticks enable discrete limit order placement, while coarser swap ticks aggregate order flow and organize liquidity movement. This tick hierarchy empowers liquidity providers with granular control while maintaining gas efficiency at scale.

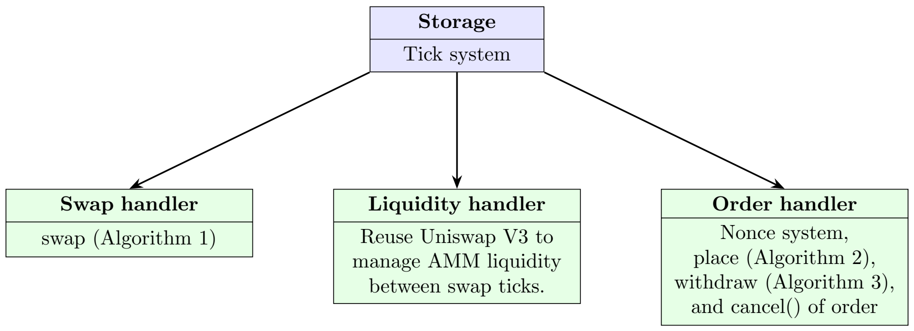
A key innovation in Monday Trade is the lazy crossing mechanism, a novel execution strategy that defers computation and storage writes during tick traversal. Rather than eagerly updating every crossed order tick during a swap, the system delays those updates until new state changes are triggered (e.g., by order placement or withdrawal). This significantly reduces gas overhead without sacrificing correctness or execution determinism, enabling practical support for high-resolution price grids.

The protocol modularizes execution into isolated components: order handler, swap engine, and liquidity manager. Each of these is equipped with nonce-tracking and state reconciliation logic to ensure safety and consistency across asynchronous interactions. Combined with a tick-aligned nonce system, this architecture enforces predictable order settlement, enables cross-tick accounting, and preserves economic soundness even under adversarial conditions.

Building upon Uniswap V3's (Adams et al., 2021) liquidity primitives, Monday Trade maintains tight synchronization between AMM pricing and order book boundaries, preserving arbitrage bounds and enabling robust execution in fast-moving markets. Its extensible core design provides a foundation for advanced financial primitives without compromising performance or decentralization.

2 Architecture

The main architecture of Monday Trade is in Figure 1.



Storage
Tick system
Swap handler
swap (Algorithm 1)
Liquidity handler
Reuse Uniswap V3 to manage AMM liquidity between swap ticks.
Order handler
Nonce system, place (Algorithm 2), withdraw (Algorithm 3), and cancel() of order

Figure 1: Architecture

The storage is inherited from Uniswap V3. The tick system consists of two sets of ticks: order tick and swap ticks. The tick spacing of order tick is denser than that of swap tick. The swap tick inherits its functionality from Uniswap V3 and it manages order ticks. The liquidity handler reuses Uniswap V3 to manage the range liquidity, for example the calculation of the active liquidity (Definition 6). The swap handler implements the lazy crossing algorithm (Algorithm 1), which is a gas-efficient way to fill orders. This helps to support finer order tick size while maintaining the gas consumption low. The order handler manages the placement, withdrawal, and cancellation of orders. Its nonce system works together with the swap handler to keep track of what orders are filled. Moreover, the nonce system defines which orders are withdrawable or cancelable.

The organization of this paper is as follows. We introduce the tick system in Section 3, liquidity handler in Section 4, the swap handler in Section 5, and the order handler in Section 6.

3 Tick system

Let $P = (0, \infty)$ be the price axis of the price of token X in token Y. Let $\underline{z} < \overline{z} \in \mathbb{Z}$, the set of integers.

Definition 1 (Order tick). *The order tick system with the spacing parameter $m \in \mathbb{N}^+$ is a map*

$$\begin{aligned} \mathcal{T}_m : [\underline{z}, \overline{z}] \cap m\mathbb{Z} &\to P \\ i &\mapsto 1.0001^i \end{aligned}$$

The domain of $\mathcal{T}_m$ is called the order ticks.

If an LP places an order consisting of token X/Y at the order tick i/j, he is willing to sell/buy X at the price $\mathcal{T}_m(i)/\mathcal{T}_m(j)$. It follows that

$$\boxed{\text{Orders consisting of X must lie to the right of that of Y, i.e. } j < i,}$$

as orders violating it would have been filled. Let

$$\begin{aligned} I &:= \min\{i \mid \text{orders on } i \text{ consist of token X only}\}, \\ J &:= \max\{j \mid \text{orders on } j \text{ consist of token Y only}\}. \end{aligned}$$

Note that $J < I$.

Definition 2 (Bid/ask price)**.** *The bid/ask price p_b/p_a of token* X *is defined as $\mathcal{T}_m(J)/\mathcal{T}_m(I)$.*

The order tick system supports the following functions to keep track of orders on each tick.

Definition 3.

$$\begin{aligned} x : [\underline{z}, \overline{z}] \cap m\mathbb{Z} &\to [0, \infty) \\ i &\mapsto \text{total amount of X token placed on } i \\ \text{need\_y} : [\underline{z}, \overline{z}] \cap m\mathbb{Z} &\to [0, \infty) \\ i &\mapsto \mathcal{T}_m(i)x(i) \\ y : [\underline{z}, \overline{z}] \cap m\mathbb{Z} &\to [0, \infty) \\ i &\mapsto \text{total amount of Y token placed on } i \\ \text{need\_x} : [\underline{z}, \overline{z}] \cap m\mathbb{Z} &\to [0, \infty) \\ i &\mapsto \frac{y(i)}{\mathcal{T}_m(i)} \end{aligned}$$

A denser order tick spacing, i.e. smaller m, provides fine control for LPs. However, the gas consumption for a trade that crosses many order ticks will be high. To improve gas efficiency, we introduce the swap tick system, which is another tick system with a coarser spacing.

Definition 4 (Swap tick)**.** *Let $\mathcal{T}_m$ be an order tick system. The swap tick system with the spacing parameter $n > m$ associated with $\mathcal{T}_m$ is a map $\mathcal{S}_n : [\underline{z}, \overline{z}] \cap n\mathbb{Z} \to P$ defined as follows.*

$$\mathcal{S}_n := \mathcal{T}_n.$$

The domain of $\mathcal{S}_n$ is called the swap ticks.

Remark. *Any swap tick i that is a multiple of* $\operatorname{lcm}(m, n)$ *is also an order tick. In particular, if $m|n$, each swap tick is an order tick.*

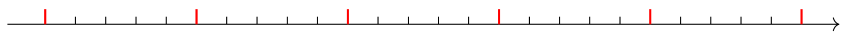


Figure 2: Swap tick system $\mathcal{S}_n$ with $n = 5m$.

Every swap tick keeps track of the total orders placed on the order ticks between it (inclusive) and the previous/next swap tick. It also keeps track of the amount of token required to fill all the orders of which it keeps track.

Definition 5 (Swap tick bookkeeping functions without AMM liquidity)**.** *Let i be the current swap tick.*

$$\begin{aligned}
order\_X : [\underline{z}, \overline{z}] \cap n\mathbb{Z} &\to [0, \infty) \\
i &\mapsto \sum_{\substack{i \le j < \min\{i+n, \overline{z}+1\} \\ m \mid j}} x(j) \\
order\_need\_Y : [\underline{z}, \overline{z}] \cap n\mathbb{Z} &\to [0, \infty) \\
i &\mapsto \sum_{\substack{i \le j < \min\{i+n, \overline{z}+1\} \\ m \mid j}} \text{need\_y}(j) \\
order\_Y : [\underline{z}, \overline{z}] \cap n\mathbb{Z} &\to [0, \infty) \\
i &\mapsto \sum_{\substack{\max\{i-n, \underline{z}-1\} < j \le i \\ m \mid j}} y(j) \\
order\_need\_X : [\underline{z}, \overline{z}] \cap n\mathbb{Z} &\to [0, \infty) \\
i &\mapsto \sum_{\substack{\max\{i-n, \underline{z}-1\} < j \le i \\ m \mid j}} \text{need\_x}(j).
\end{aligned}$$

Remark. *$order\_X(i)$ is the amount of X stored between $[i, i+n)$*

4 Liquidity handler

Any constant function AMM quotes a current price p_c of X in Y by implicit differentiation

$$p_c = -\frac{dy}{dx} = \frac{\partial F/\partial x}{\partial F/\partial y}, \tag{1}$$

where $F(x, y)$ is the invariant curve. For concentrated liquidity AMM, Equation (1) holds true when applied to virtual reserves.

An AMM liquidity over price range $(0, \infty)$ contains both X and Y across $(0, \infty)$, which seemingly contradicts our requirement in Section 3 that token X lie to the right of token Y on the price axis. The following proposition resolves this contradiction.

Proposition 1 (Jiang et al. (2025))**.** *For an AMM with current price p_c and liquidity l over $(0, \infty)$, if a trade moves the price to p, then for any interval $(p_1, p_2) \ni p_c, p$, l can be decomposed as three range liquidities with value l over $(0, p_1)$, (p_1, p_2), and (p_2, ∞)*[1]

For our purpose, we need the following corollary.

[1] l over $(0, p_1)$, (p_1, p_2), and (p_2, ∞) means $\sqrt{p_1} l$ amount of Y, $\left(\sqrt{\frac{1}{p_c}} - \sqrt{\frac{1}{p_2}}\right) l$ amount of X and $\left(\sqrt{p_c} - \sqrt{p_1}\right) l$ amount of Y, and $\sqrt{\frac{1}{p_2}} l$ amount of X, respectively.

Corollary 1. *An AMM liquidity l over $(p_1, p_2) \ni p_c$ can be decomposed to l over (p_1, p_c) and (p_c, p_2).*

Proof. Let $p \to p_c$ in Proposition 1. $\square$

Corollary 1 says that a range liquidity containing the AMM current price p_c is equivalent to two range liquidities sitting next to p_c such that the liquidity to the left/right of p_c contains only Y/X.

Because Monday Trade supports order tick

Theorem 1. *If p_c exists, then $p_b \leq p_c \leq p_a$ but the two equalities cannot hold simultaneously.*

Proof. Suppose that there is a background liquidity l over $(0, \infty)$ and an order at p_b consisting of $y > \sqrt{\frac{p_b}{p_c}}(\sqrt{p_b} - \sqrt{p_c})l$ amount of Y. Suppose for contradiction that $p_c < p_b < p_a$. WLOG, suppose that there exists no other order nor range liquidity in $(p_c, p_b]$.

1. Use $(\sqrt{p_b} - \sqrt{p_c})l$ amount of Y to buy $\frac{\sqrt{p_b}-\sqrt{p_c}}{\sqrt{p_b p_c}}l$ amount of X. This pushes the AMM price to p_b.
2. Use $\frac{\sqrt{p_b}-\sqrt{p_c}}{\sqrt{p_b p_c}}l$ amount of X to buy Y. This fills the bid order at p_b and we obtain $\sqrt{\frac{p_b}{p_c}}(\sqrt{p_b} - \sqrt{p_c})l$ amount of Y.
3. We have earned $\frac{(\sqrt{p_b}-\sqrt{p_c})^2}{\sqrt{p_c}}l$ amount of Y, an arbitrage.

Similarly we can show that $p_c \leq p_a$. $\square$

Theorem 1 says that the current price of the AMM must lie between the bid and ask prices of the order book. Otherwise there will be arbitrage opportunity within the system.

In Definition 5, *order_X* and *order_Y* are defined as functions on the swap tick and can be evaluated against any input. One might attempt to update their definition to include range liquidity so that each swap tick keeps track of the total amount of token needed to cross it. This is infeasible because if a background full range liquidity is added, we need to update all swap ticks. To efficiently manage both order and range liquidities, we require that

$$\boxed{\text{Range liquidity be placed between any two swap ticks.}}$$

This allows us to use the concept of active liquidity in Uniswap V3 to manage AMM liquidity. Note that Proposition 1 implies that range liquidity is additive, i.e. if there are l_1 over (p_1, p_2) and l_2 over (p_3, p_4), where $p_1 < p_3 < p_2 < p_4$, then the liquidity amount in (p_3, p_2) is $l_1 + l_2$. This allows us to define the following.

Definition 6 (Active liquidity). *Let the current AMM price be p_c and the nearest two swap ticks to p_c be i and $i+n$. The current liquidity l is defined as the sum*

$$l \coloneqq \sum_{l' \text{ covers } (\mathcal{T}_n(i), \mathcal{T}_n(i+n))} l',$$

where l' covers $(\mathcal{T}_n(i), \mathcal{T}_n(i+n))$ means the range of l' contains $(\mathcal{T}_n(i), \mathcal{T}_n(i+n))$. If the AMM current price doesn't exist, l is defined to be 0.

Remark. *Note that both i and l are functions of p_c.*

Definition 7. *Let p_c the current price, i and $i+n$ be two nearest swap ticks to p_c, and l the current liquidity. The following two functions $amm\_X, amm\_Y : (0,\infty) \cup \{\emptyset\} \to [0,\infty)$ are defined as follows*

$$\begin{aligned} amm\_X(p_c) &\coloneqq \left(\sqrt{\frac{1}{p_c}} - \sqrt{\frac{1}{\mathcal{T}_n(i+n)}}\right) l, \\ amm\_Y(p_c) &\coloneqq \left(\sqrt{p_c} - \sqrt{\mathcal{T}_n(i)}\right) l \end{aligned}$$

and

$$\begin{aligned} amm\_X(\emptyset) &\coloneqq 0, \\ amm\_Y(\emptyset) &\coloneqq 0. \end{aligned}$$

Note that Definition 7 only defines the amount of token needed to cross the swap ticks surrounding p_c. Once these ticks are crossed, Uniswap V3 updates the current liquidity l on the fly and in turn *amm_X* and *amm_Y* are correctly re-calculated.

5 Swap handler

In this section, we introduce the following lazy tick crossing mechanism:

> If a swap is to fill all the order ticks between two swap ticks, we only update the information of these two swap ticks. The update of the order ticks in between is postponed till new orders are placed on them or tokens are withdrawn from them.

The detail for lazy tick crossing for buying token X is given in Algorithm 1 and Figure 3. The procedure for Y is similar.

A crossing of a swap tick implies the crossing of all order ticks associated with it, but only the swap tick is updated in lazy crossing. In Section 6, we will use nonce to rectify this. For the gas saving effect of lazy crossing, consider a swap starting from the tick i that crosses N consecutive order ticks without lazy tick crossing, the actual tick crossing is

$$\sum_{\substack{i \le j < i+mN \\ n \mid j}} 1 = \left\lceil \frac{i+mN}{n} \right\rceil - \left\lceil \frac{i}{n} \right\rceil$$

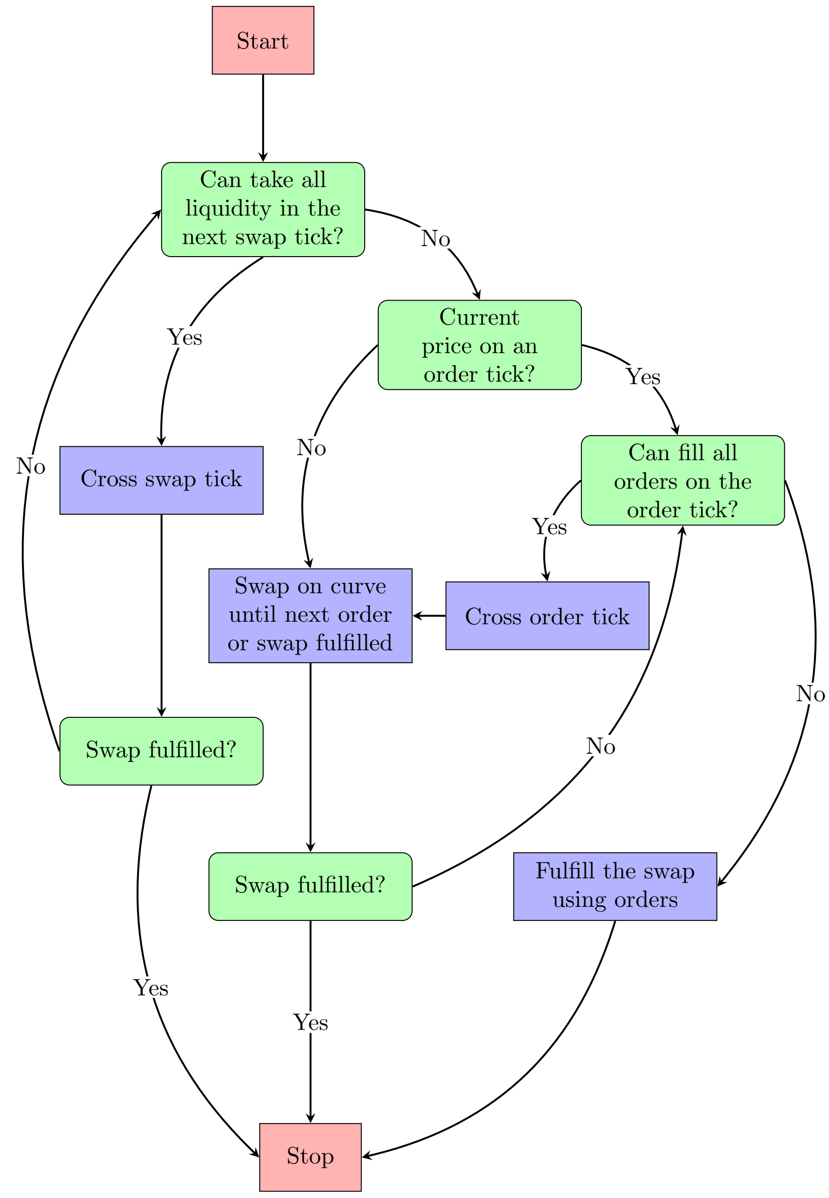

Start
Can take all liquidity in the next swap tick?
No
Yes
Current price on an order tick?
Yes
No
Cross swap tick
Can fill all orders on the order tick?
Yes
Swap on curve until next order or swap fulfilled
Cross order tick
No
Swap fulfilled?
No
Swap fulfilled?
Fulfill the swap using orders
Yes
Yes
Stop

Figure 3: Lazy tick crossing flowchart

Algorithm 1: Lazy tick crossing for buying X

```
Data: $i$: initial tick, $t$: input token amount
$output \leftarrow 0$ ;
$current\_tick \leftarrow i$ ;
if $current\_tick$ is a swap tick then
    $next\_tick \leftarrow i$ ;
else
    $next\_tick \leftarrow next\_swap\_tick(current\_tick)$ ;
end
while $t \geq next\_tick.need\_Y$ do
    $output \leftarrow output + next\_tick.X$ ;
    $t \leftarrow t - next\_tick.need\_Y$ ;
    $next\_tick.X \leftarrow 0$ ;
    $next\_tick.need\_Y \leftarrow 0$ ;
    $next\_tick.nonce\_x \leftarrow next\_tick.nonce\_x + 1$ ;
    $current\_tick \leftarrow next\_tick$ ;
    $next\_tick \leftarrow next\_swap\_tick(current\_tick)$ ;
end
if $t > 0$ then
    foreach $i \in \{current\_tick < i < next\_tick \mid m \text{ divides } i\}$ do
        if $t \geq i.need\_y$ then
            $output \leftarrow output + i.x$ ;
            $t \leftarrow t - i.need\_y$ ;
            $next\_tick.X \leftarrow next\_tick.X - i.x$ ;
            $next\_tick.need\_Y \leftarrow next\_tick.need\_Y - i.need\_y$ ;
            $i.x \leftarrow 0$;
            $i.need\_y \leftarrow 0$ ;
            $i.nonce\_x \leftarrow i.nonce\_x + 1$ ;
        else
            $output \leftarrow output + \frac{t}{\mathcal{T}_m(i)}$ ;
            $next\_tick.X \leftarrow next\_tick.X - \frac{t}{\mathcal{T}_m(i)}$ ;
            $next\_tick.need\_Y \leftarrow next\_tick.need\_Y - t$ ;
            $i.x \leftarrow i.x - \frac{t}{\mathcal{T}_m(i)}$ ;
            $i.need\_y \leftarrow i.need\_y - t$ ;
            break ;
        end
    end
end
return $output$
```

swap tick crossings plus

$$\sum_{\substack{\max\left\{i-1,\left\lfloor\frac{i+mN-1}{n}\right\rfloor n\right\}<j<i+mN\\ m\,\mid\, j}} 1$$

$$=\begin{cases}\frac{i}{m}+N-\left\lfloor\frac{\left\lfloor\frac{i+mN-1}{n}\right\rfloor n}{m}\right\rfloor-1, & i<\left\lfloor\frac{i+mN-1}{n}\right\rfloor n\\ N, & otherwise\end{cases}$$

order tick crossings.

A concrete example is given below.

Example 1. *Suppose $m = 3$, $n = 300$, $i = 0$, and $N = 101$. In this case,*

$$\left\lceil\frac{0+3\times 101}{300}\right\rceil-\left\lceil\frac{0}{300}\right\rceil=2$$

swap ticks are crossed and

$$\frac{0}{3}+101-\left\lfloor\frac{\left\lfloor\frac{0+3\times101-1}{300}\right\rfloor\times 300}{3}\right\rfloor-1=0$$

order tick is crossed, a 98% save on tick crossing.

6 Order handler

We introduce the following nonce (Number used ONCE) system to maintain the correctness of the system.

Definition 8 (Nonce)**.** *Each order and swap tick keeps track of two nonces: nonce_x and nonce_y. If the tick i consists of token X and a swap depletes all X on i, then i.nonce_x is incremented by 1. Similarly for Y.*

Since a crossing of a swap tick implies the crossing of all order ticks associated with it, the nonce of the swap tick should not be larger than that of its associated order ticks. However, lazy crossing (Algorithm 1) violates this as it only updates swap tick's nonce. We call it the nonce discrepancy.

Definition 9 (Nonce discrepancy)**.** *The system is said to be in nonce discrepancy if there exists one swap tick whose nonce is larger than that of its associated order ticks.*

To rectify this, we update the nonces of the corresponding order ticks at the time of order placement (Algorithm 2 and Figure 4).

The withdrawal mechanism (Algorithm 3 and Figure 5) implies the following desired property.

$$\boxed{\text{Fully filled orders buying X/Y are withdrawable in X/Y.}}$$

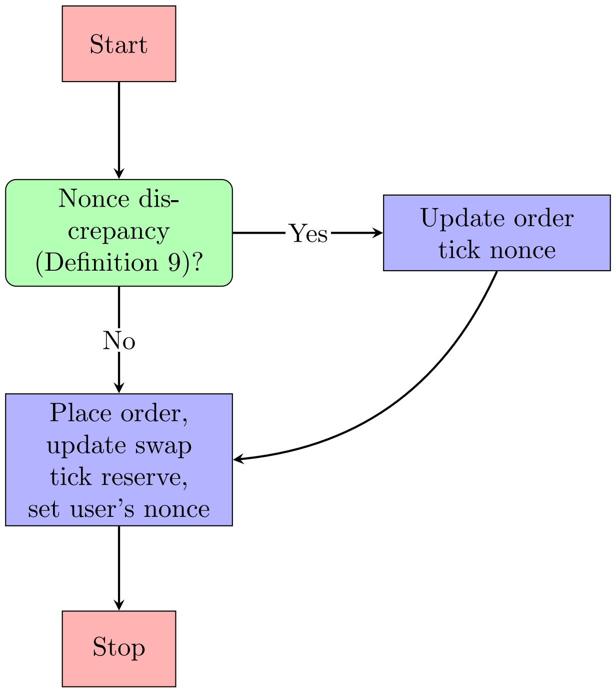

Start
Nonce discrepancy (Definition 9)?
Yes
Update order tick nonce
No
Place order, update swap tick reserve, set user's nonce
Stop

Figure 4: Order placement

Algorithm 2: Order placement

```
Data: $i$: tick to place order, $t$: token amount, $placed\_x$: True if the
      input token is X
if $nonce\ discrepancy$ then
    update order tick's nonce ;
end
if $placed\_x$ then
    $i.x \leftarrow i.x + t$ ;
    $i.need\_y \leftarrow i.need\_y + \mathcal{T}_m(i)t$ ;
    if $i$ is a swap tick then
        $i.X \leftarrow i.X + t$ ;
        $i.need\_Y \leftarrow i.need\_Y + \mathcal{T}_m(i)t$ ;
    else
        $next\_swap\_tick.X \leftarrow next\_swap\_tick.X + t$ ;
        $next\_swap\_tick.need\_Y \leftarrow next\_swap\_tick.need\_Y + \mathcal{T}_m(i)t$ ;
    end
    $user.nonce \leftarrow i.nonce\_x$ ;
else
    $i.y \leftarrow i.y + t$ ;
    $i.need\_x \leftarrow i.need\_x + \frac{t}{\mathcal{T}_m(i)}$ ;
    if $i$ is a swap tick then
        $i.Y \leftarrow i.Y + t$ ;
        $i.need\_X \leftarrow i.need\_X + \frac{t}{\mathcal{T}_m(i)}$ ;
    else
        $prev\_swap\_tick.Y \leftarrow prev\_swap\_tick.Y + t$ ;
        $prev\_swap\_tick.need\_X \leftarrow prev\_swap\_tick.need\_X + \frac{t}{\mathcal{T}_m(i)}$ ;
    end
    $user.nonce \leftarrow i.nonce\_y$ ;
end
```

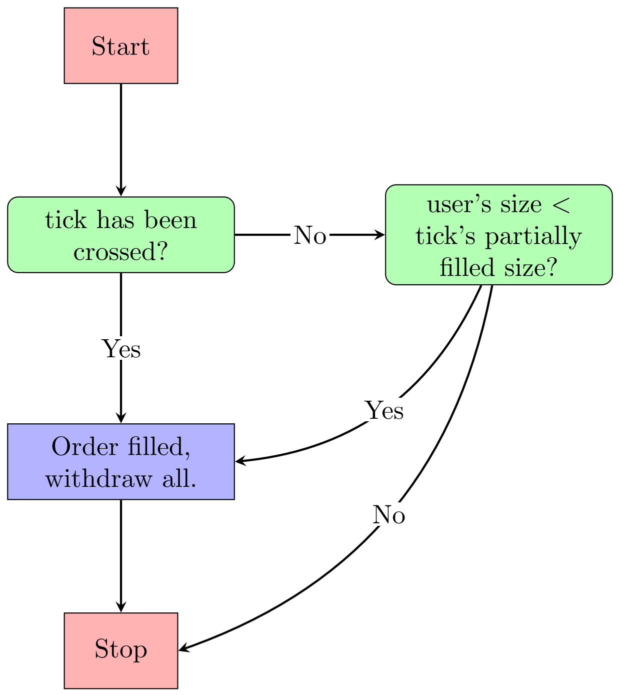

Start
tick has been crossed?
No
user's size < tick's partially filled size?
Yes
Yes
Order filled, withdraw all.
No
Stop

Figure 5: Order withdrawal

Algorithm 3: Withdrawal

```
Data: $i$: tick to withdraw from, $t$: deposited token amount, $placed\_x$:
      True if the deposited token is X
if $placed\_x$ then
    if tick $i$ has been crossed then
        return $\mathcal{T}_m(i) \cdot t$ amount of token $Y$ ;
    else if $t < partially\_filled\_size$ then // tick is not crossed
     but the order has been filled.
        $partially\_filled\_size \leftarrow partially\_filled\_size - t$ ;
        return $\mathcal{T}_m(i) \cdot t$ amount of token $Y$ ;
    end
else
    if tick $i$ has been crossed then
        return $\frac{t}{\mathcal{T}_m(i)}$ amount of token $X$ ;
    else if $t < partially\_filled\_size$ then
        $partially\_filled\_size \leftarrow partially\_filled\_size - t$ ;
        return $\frac{t}{\mathcal{T}_m(i)}$ amount of token $X$ ;
    end
end
```

Indeed, for an order selling X placed at time t_0 at tick i, the user's nonce is set to $i.nonce\_x$. If tick i or the swap tick managing it is crossed at time $t_1 > t_0$, $i.nonce\_x$ will be incremented and the Y token obtained will be stored in the pool's contract and the user can withdraw it at any time.

Order handler also supports order cancellation (Algorithm 4and Figure 6). The relation between withdrawal of the cancellation is as follows:

1. Only fully filled orders can be withdrawn.
2. Partially filled or untouched orders can be canceled.
3. Sometimes, a fully filled order may also be canceled (Example 2).

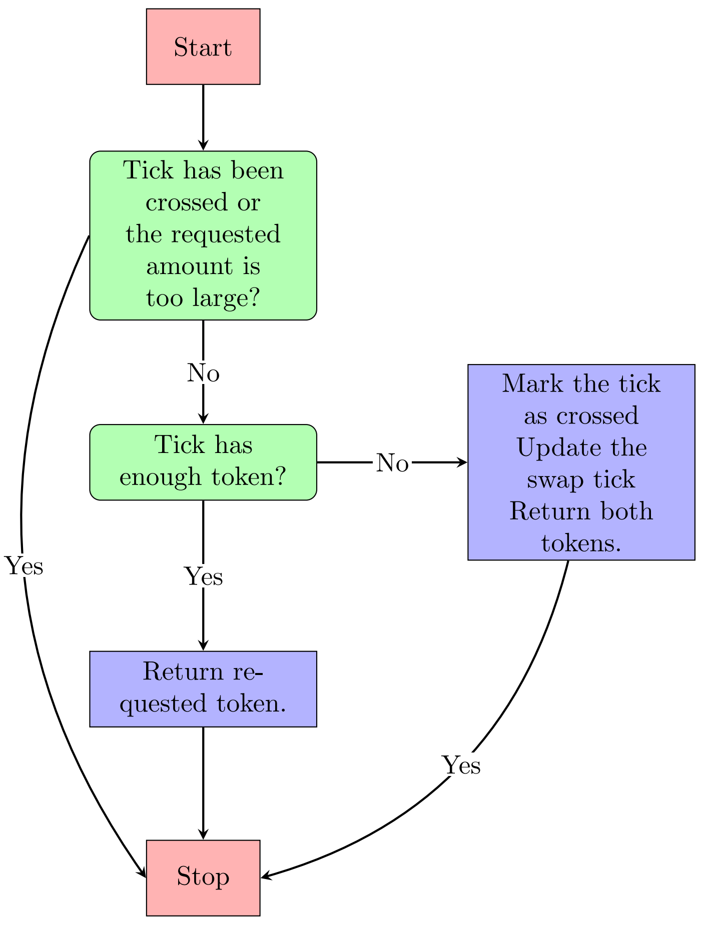

Start
Tick has been crossed or the requested amount is too large?
No
Yes
Tick has enough token?
No
Mark the tick as crossed Update the swap tick Return both tokens.
Yes
Return requested token.
Yes
Stop

Figure 6: Order cancellation

Example 2. *User A placed* 100 *X at price* 10*, user B placed* 100 *X at the same price. User C bought* 100 *X at this price. Now, A may choose to withdraw as*

Algorithm 4: Cancellation

```
Data: $i$: tick to cancel at, $t$: amount to cancel, $placed\_x$: True if the
      deposited token is X
if $placed\_x$ then
    if $tick\ i\ has\ been\ crossed$ or $tick\_size + partial\_filled\_size < t$ then
        return;
    else if $t \leq tick\_size$ then// normal cancellation.
        $partially\_filled\_size \leftarrow partially\_filled\_size - t$ ;
        Update the corresponding swap tick ;
        return $t\ amount\ of\ token\ X$ ;
    else // Cancellation of a partially filled order.
        $x \leftarrow tick\_size$ ;
        $y \leftarrow (t - tick\_size) \cdot \mathcal{T}_m(i)$ ;
        $tick\_size \leftarrow 0$ ;
        $partially\_filled\_size \leftarrow 0$ ;
        $i.nonce\_x \leftarrow i.nonce\_x + 1$ ;        // mark the order tick as
         filled as the remaining order is fully filled.
        Update the corresponding swap tick ;
        return $x\ amount\ of\ token\ X\ and\ y\ amount\ of\ token\ Y$ ;
    end
else
    if $tick\ i\ has\ been\ crossed$ or $tick\_size + partial\_filled\_size < t$ then
        return;
    else if $t \leq tick\_size$ then
        $partially\_filled\_size \leftarrow partially\_filled\_size - t$ ;
        Update the corresponding swap tick ;
        return $t\ amount\ of\ token\ Y$ ;
    else
        $y \leftarrow tick\_size$ ;
        $x \leftarrow \frac{t - tick\_size}{\mathcal{T}_m(i)}$ ;
        $tick\_size \leftarrow 0$ ;
        $partially\_filled\_size \leftarrow 0$ ;
        $i.nonce\_y \leftarrow i.nonce\_y + 1$ ;
        Update the corresponding swap tick ;
        return $x\ amount\ of\ token\ X\ and\ y\ amount\ of\ token\ Y$ ;
    end
end
```

his order is fully filled and he will obtain 1000 *Y. A may also choose to cancel his order and he will gets his* 100 *X back. This will make the tick crossed and B can only withdraw, not cancel.*

References

[1] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson. *Uniswap v3 Core.* Tech. rep. Uniswap, 2021. URL: `https://app.uniswap.org/whitepaper-v3.pdf`.

[2] R. Jiang, L. Wen, Y. Cao, and Y. Ding. "Chasing price drains liquidity". In: *Mathematical Research for Blockchain Economy. 6th International Conference MARBLE 2025, Athens, Greece.* Ed. by W. Knottenbelt, S. Leonardos, A. Goharshady, and P. Pardalos. Lecture Notes in Operations Research. Springer, 2025.